Amendments to the Claims:

The following claims will replace all prior versions, and listing, of claims in this application:

1. (Original) An apparatus for wrapping existing procedure oriented program into component based system, comprising:

a code analyzing portion for extracting information necessary for program analysis in source program or codes implemented with source procedural language;

a business logic identifying portion for identifying a portion of very high probability of reuse using the information necessary for program analysis extracted in the code analyzing portion; and

a component wrapper generating portion for automatically generating the codes for wrapping the program workflow which includes business logic identified in the business logic identifying portion.

2. (Original) The apparatus as claimed in claim 1, wherein the component wrapper generating portion comprises:

a component framework for reusing existing system as a component;

a legacy frame work which is a framework of system to be associated with the component framework; and

an intermediate framework for linking the component framework with the legacy frame work, and capturing screen information which is input/output to/from the legacy framework, thereby automatically inserting or extracting information.

3. (Currently amended) The apparatus as claimed in claim 2, wherein the intermediate framework comprises:

a program scheduler having ~~voyage~~ navigation information and interaction relationship between programs, and having schedule information about whether a plurality of screens are for input or for output;

a meta-data pool for storing meta-information for the screens of programs included in a pre-registered workflow;

a record handler for analyzing the ~~commond~~ command required by the component framework, obtaining the meta information of input/output data from the meta-data pool,

thereby finding which are the screens entered from the present existing system and which are the input/output data corresponding to the screens, and for transferring the input/output data; and

a record adapter for receiving input screen from the legacy component, differentiating the data associated with the input/output from the information for display of screen only, and providing it to the record handler.

4. (Original) The apparatus as claimed in claim 3, wherein the record adapter stores temporarily the information of legacy component, and transforms different characters of ASCII, EBCDIC, etc.

5. (Original) The apparatus as claimed in claim 3, wherein the program scheduler stores the information about how the program is used for any usage of input, output or input/output usage.

6. (Original) A method for wrapping existing procedure oriented program into component based system, comprising the steps of:

extracting information necessary for program analysis in source program or codes implemented with source procedural language;

identifying a portion of very high probability of reuse using the information necessary for program analysis extracted in the code analyzing portion; and

generating automatically the codes for wrapping program workflow which include business logic identified in the business logic identifying portion.

7. (Currently amended) The method as claimed in claim 6, wherein the step of identifying comprises the steps of:

calculating the fitting index of user requirement using weighting value of the constituent elements inputted by user depending on a scale of each module in order to express business type to be identified;

a) determining ~~whether~~ where the calculated fitting index is the largest, ~~if the fitting index is the largest~~, then searching the flows within program for executing module in the program including the module where the fitting index is the largest,

b) searching input/output variables based on variables associated with screen decoration having the direct relations with user;

identifying automatically variables necessary for constraint condition and interface using input/output variables and flows (path) within the searched program; and

defining the variables to be constraint condition and variables to be interface using the identified variables, to generate the code for the wrapping.

8. (Original) The method as claimed in claim 6, wherein the calculation of the fitting index in the step of calculating the fitting index, lies in that it calculates fitness (fitting index) about the user requirement in Top-Down method which searches form large portion to small portion in scale.

9. (Original) The method as claimed in claim 7, wherein the constraint condition consists of control variables necessary to obtain the flow for executing the module of the desired business logic, and wherein the interface consists of variables utilized in input/output portion of data.

10. (Currently amended) The method as claimed in claim 7, wherein in the step of searching the flows within the program and the input/output variables, the step of searching the flows within the program comprises the steps of:

collecting the flow information between paragraphs and the information about the condition thereof by the unit of modules and paragraphs in order to adjust the inferior information of the program systematically when the programs for the detailed analysis of program, the least modules having business logics, and the paragraph candidates are known, searching call relations between the paragraphs using function call statement for searching call relations between the modules;

eliminating redundancy or recursive portions of the paragraph calls for inclusive call relations, ~~if there are inclusive call relations,~~ then reconstructing the paragraph call relations; identifying the flow of the program taking account of only unstructured statement sentence of the flow information between the paragraphs for searching the paragraph flow by the unstructured statement; and

generating call relation tree using the call relation information of the paragraph acquired and the program flow information of the unstructured sentence.

11. (Original) The method as claimed in claim 7, wherein in the step of searching call relations between the paragraphs, the function call statement utilizes CALL sentence

and PERFORM sentence in COBOL.

12. (Original) The method as claimed in claim 10, wherein in the step of identifying the flow of the program, the unstructured statement utilizes at least one of GO TO sentence, CONTINUE sentence and BREAK sentence in COBOL.

13. (Currently amended) The method as claimed in claim 7, wherein in the step of searching the flows within program and the input/output variables, the step of searching the input/output variables comprises the steps of:

analyzing the screen information of each variable and field, by analyzing the input/output variables which exist in the program having business logic to be reused, the information about user interface or forms for expressing a screen;

determining whether or not a field exists in the analyzed screen information;

discriminating~~, if the field exists in the analyzed screen information,~~ whether the field is a portion for input/output of actual data or only for decoration of screen when the field exists in the analyzed screen information; and

a) registering, if the field is for input/output (I/O) field, the field as an input/output variable since the field is used as input/output variable,

b) registering, if the field is not for the portion for input/output, the field as meta data since the field is used as decoration of the screen.

14. (Currently amended)The method as claimed in claim 7, wherein the step of identifying automatically variables necessary for constraint condition and interface, comprises the steps of:

selecting Unique Path having the workflow that user wants in the generated tree;

checking whether there exist Critical variable ~~such as variables deciding~~ that decide the paragraph flow or input/output in the designated workflow;

tracking, ~~if there exists the critical variable,~~ for the critical variable, the list of variables affecting the critical variable using impact analysis, or tracking, [[if]] for the variables ~~are those~~ transferred between programs, continuously the calling programs or the called programs and identifying the usage of variables;

discriminating whether the identified variables are those of determining workflow path or those utilized as a constraint condition; and

a) [[if]] for the identified variables ~~are~~ used as a control variable, adding them to the list of the control variables,

b) [[if]] for the identified variables [[are]] used as a constraint condition, adding them to the list of the a constraint condition.

15. (Currently amended) A recording medium capable of being read by a digital processing apparatus, in which programs capable of being executed by the digital processing apparatus are implemented by types so as to perform a method for wrapping existing procedure oriented program into component based system, wherein the method comprises the steps of:

extracting information necessary for program analysis in source program or codes implemented with source procedural language;

identifying a portion of very high probability of reuse using the information necessary for program analysis extracted in the code analyzing portion; and

generating automatically the codes for wrapping program workflow which includes business logic identified in the business logic identifying portion,

the step of identifying comprising the steps of:

calculating the fitting index of user requirement using weighting value of the constituent elements inputted by user depending on a scale of each module in order to express business type to be identified;

a) determining whether the calculated fitting index is the largest~~, if the fitting index is the largest,~~ then searching, for the largest fitting index, the flows within program for executing module in the program including the module where the fitting index is the largest,

b) searching input/output variables based on variables associated with screen decoration having the direct relations with user;

identifying automatically variables necessary for constraint condition and interface using input/output variables and flows (path) within the searched program; and

defining the variables to be constraint condition and variables to be interface using the identified variables, to generate the code for the wrapping.